JON

Network Working Group
Request for Comments: 1058

C. Hedrick
Rutgers University
June 1988

Routing Information Protocol

Status of this Memo

This RFC describes an existing protocol for exchanging routing information among gateways and other hosts. It is intended to be used as a basis for developing gateway software for use in the Internet community. Distribution of this memo is unlimited.

Table of Contents

1. Introduction 2
   1.1. Limitations of the protocol 4
   1.2. Organization of this document 4
2. Distance Vector Algorithms 5
   2.1. Dealing with changes in topology 11
   2.2. Preventing instability 12
      2.2.1. Split horizon 14
      2.2.2. Triggered updates 15
3. Specifications for the protocol 16
   3.1. Message formats 18
   3.2. Addressing considerations 20
   3.3. Timers 23
   3.4. Input processing 24
      3.4.1. Request 25
      3.4.2. Response 26
   3.5. Output Processing 28
   3.6. Compatibility 31
4. Control functions 31

Overview

This memo is intended to do the following things:

- Document a protocol and algorithms that are currently in wide use for routing, but which have never been formally documented.

- Specify some improvements in the algorithms which will improve stability of the routes in large networks. These improvements do not introduce any incompatibility with existing implementations. They are to be incorporated into

all implementations of this protocol.

- Suggest some optional features to allow greater configurability and control. These features were developed specifically to solve problems that have shown up in actual use by the NSFnet community. However, they should have more general utility.

The Routing Information Protocol (RIP) described here is loosely based on the program "routed", distributed with the 4.3 Berkeley Software Distribution. However, there are several other implementations of what is supposed to be the same protocol. Unfortunately, these various implementations disagree in various details. The specifications here represent a combination of features taken from various implementations. We believe that a program designed according to this document will interoperate with routed, and with all other implementations of RIP of which we are aware.

Note that this description adopts a different view than most existing implementations about when metrics should be incremented. By making a corresponding change in the metric used for a local network, we have retained compatibility with other existing implementations. See section 3.6 for details on this issue.

1. Introduction

This memo describes one protocol in a series of routing protocols based on the Bellman-Ford (or distance vector) algorithm. This algorithm has been used for routing computations in computer networks since the early days of the ARPANET. The particular packet formats and protocol described here are based on the program "routed", which is included with the Berkeley distribution of Unix. It has become a de facto standard for exchange of routing information among gateways and hosts. It is implemented for this purpose by most commercial vendors of IP gateways. Note, however, that many of these vendors have their own protocols which are used among their own gateways.

This protocol is most useful as an "interior gateway protocol". In a nationwide network such as the current Internet, it is very unlikely that a single routing protocol will used for the whole network. Rather, the network will be organized as a collection of "autonomous systems". An autonomous system will in general be administered by a single entity, or at least will have some reasonable degree of technical and administrative control. Each autonomous system will have its own routing technology. This may well be different for different autonomous systems. The routing protocol used within an autonomous system is referred to as an interior gateway protocol, or "IGP". A separate protocol is used to interface among the autonomous

systems. The earliest such protocol, still used in the Internet, is "EGP" (exterior gateway protocol). Such protocols are now usually referred to as inter-AS routing protocols. RIP was designed to work with moderate-size networks using reasonably homogeneous technology. Thus it is suitable as an IGP for many campuses and for regional networks using serial lines whose speeds do not vary widely. It is not intended for use in more complex environments. For more information on the context into which RIP is expected to fit, see Braden and Postel [3].

RIP is one of a class of algorithms known as "distance vector algorithms". The earliest description of this class of algorithms known to the author is in Ford and Fulkerson [6]. Because of this, they are sometimes known as Ford-Fulkerson algorithms. The term Bellman-Ford is also used. It comes from the fact that the formulation is based on Bellman's equation, the basis of "dynamic programming". (For a standard introduction to this area, see [1].) The presentation in this document is closely based on [2]. This text contains an introduction to the mathematics of routing algorithms. It describes and justifies several variants of the algorithm presented here, as well as a number of other related algorithms. The basic algorithms described in this protocol were used in computer routing as early as 1969 in the ARPANET. However, the specific ancestry of this protocol is within the Xerox network protocols. The PUP protocols (see [4]) used the Gateway Information Protocol to exchange routing information. A somewhat updated version of this protocol was adopted for the Xerox Network Systems (XNS) architecture, with the name Routing Information Protocol. (See [7].) Berkeley's routed is largely the same as the Routing Information Protocol, with XNS addresses replaced by a more general address format capable of handling IP and other types of address, and with routing updates limited to one every 30 seconds. Because of this similarity, the term Routing Information Protocol (or just RIP) is used to refer to both the XNS protocol and the protocol used by routed.

RIP is intended for use within the IP-based Internet. The Internet is organized into a number of networks connected by gateways. The networks may be either point-to-point links or more complex networks such as Ethernet or the ARPANET. Hosts and gateways are presented with IP datagrams addressed to some host. Routing is the method by which the host or gateway decides where to send the datagram. It may be able to send the datagram directly to the destination, if that destination is on one of the networks that are directly connected to the host or gateway. However, the interesting case is when the destination is not directly reachable. In this case, the host or gateway attempts to send the datagram to a gateway that is nearer the destination. The goal of a routing protocol is very simple: It is to

supply the information that is needed to do routing.

1.1. Limitations of the protocol

This protocol does not solve every possible routing problem. As mentioned above, it is primary intended for use as an IGP, in reasonably homogeneous networks of moderate size. In addition, the following specific limitations should be mentioned:

- The protocol is limited to networks whose longest path involves 15 hops. The designers believe that the basic protocol design is inappropriate for larger networks. Note that this statement of the limit assumes that a cost of 1 is used for each network. This is the way RIP is normally configured. If the system administrator chooses to use larger costs, the upper bound of 15 can easily become a problem.

- The protocol depends upon "counting to infinity" to resolve certain unusual situations. (This will be explained in the next section.) If the system of networks has several hundred networks, and a routing loop was formed involving all of them, the resolution of the loop would require either much time (if the frequency of routing updates were limited) or bandwidth (if updates were sent whenever changes were detected). Such a loop would consume a large amount of network bandwidth before the loop was corrected. We believe that in realistic cases, this will not be a problem except on slow lines. Even then, the problem will be fairly unusual, since various precautions are taken that should prevent these problems in most cases.

- This protocol uses fixed "metrics" to compare alternative routes. It is not appropriate for situations where routes need to be chosen based on real-time parameters such a measured delay, reliability, or load. The obvious extensions to allow metrics of this type are likely to introduce instabilities of a sort that the protocol is not designed to handle.

1.2. Organization of this document

The main body of this document is organized into two parts, which occupy the next two sections:

2 A conceptual development and justification of distance vector algorithms in general.

3 The actual protocol description.

Each of these two sections can largely stand on its own. Section 2 attempts to give an informal presentation of the mathematical underpinnings of the algorithm. Note that the presentation follows a "spiral" method. An initial, fairly simple algorithm is described. Then refinements are added to it in successive sections. Section 3 is the actual protocol description. Except where specific references are made to section 2, it should be possible to implement RIP entirely from the specifications given in section 3.

2. Distance Vector Algorithms

Routing is the task of finding a path from a sender to a desired destination. In the IP "Catenet model" this reduces primarily to a matter of finding gateways between networks. As long as a message remains on a single network or subnet, any routing problems are solved by technology that is specific to the network. For example, the Ethernet and the ARPANET each define a way in which any sender can talk to any specified destination within that one network. IP routing comes in primarily when messages must go from a sender on one such network to a destination on a different one. In that case, the message must pass through gateways connecting the networks. If the networks are not adjacent, the message may pass through several intervening networks, and the gateways connecting them. Once the message gets to a gateway that is on the same network as the destination, that network's own technology is used to get to the destination.

Throughout this section, the term "network" is used generically to cover a single broadcast network (e.g., an Ethernet), a point to point line, or the ARPANET. The critical point is that a network is treated as a single entity by IP. Either no routing is necessary (as with a point to point line), or that routing is done in a manner that is transparent to IP, allowing IP to treat the entire network as a single fully-connected system (as with an Ethernet or the ARPANET). Note that the term "network" is used in a somewhat different way in discussions of IP addressing. A single IP network number may be assigned to a collection of networks, with "subnet" addressing being used to describe the individual networks. In effect, we are using the term "network" here to refer to subnets in cases where subnet addressing is in use.

A number of different approaches for finding routes between networks are possible. One useful way of categorizing these approaches is on the basis of the type of information the gateways need to exchange in order to be able to find routes. Distance vector algorithms are based on the exchange of only a small amount of information. Each

entity (gateway or host) that participates in the routing protocol is assumed to keep information about all of the destinations within the system. Generally, information about all entities connected to one network is summarized by a single entry, which describes the route to all destinations on that network. This summarization is possible because as far as IP is concerned, routing within a network is invisible. Each entry in this routing database includes the next gateway to which datagrams destined for the entity should be sent. In addition, it includes a "metric" measuring the total distance to the entity. Distance is a somewhat generalized concept, which may cover the time delay in getting messages to the entity, the dollar cost of sending messages to it, etc. Distance vector algorithms get their name from the fact that it is possible to compute optimal routes when the only information exchanged is the list of these distances. Furthermore, information is only exchanged among entities that are adjacent, that is, entities that share a common network.

Although routing is most commonly based on information about networks, it is sometimes necessary to keep track of the routes to individual hosts. The RIP protocol makes no formal distinction between networks and hosts. It simply describes exchange of information about destinations, which may be either networks or hosts. (Note however, that it is possible for an implementor to choose not to support host routes. See section 3.2.) In fact, the mathematical developments are most conveniently thought of in terms of routes from one host or gateway to another. When discussing the algorithm in abstract terms, it is best to think of a routing entry for a network as an abbreviation for routing entries for all of the entities connected to that network. This sort of abbreviation makes sense only because we think of networks as having no internal structure that is visible at the IP level. Thus, we will generally assign the same distance to every entity in a given network.

We said above that each entity keeps a routing database with one entry for every possible destination in the system. An actual implementation is likely to need to keep the following information about each destination:

- address: in IP implementations of these algorithms, this will be the IP address of the host or network.

- gateway: the first gateway along the route to the destination.

- interface: the physical network which must be used to reach the first gateway.

- metric: a number, indicating the distance to the

destination.

- timer: the amount of time since the entry was last updated.

In addition, various flags and other internal information will probably be included. This database is initialized with a description of the entities that are directly connected to the system. It is updated according to information received in messages from neighboring gateways.

The most important information exchanged by the hosts and gateways is that carried in update messages. Each entity that participates in the routing scheme sends update messages that describe the routing database as it currently exists in that entity. It is possible to maintain optimal routes for the entire system by using only information obtained from neighboring entities. The algorithm used for that will be described in the next section.

As we mentioned above, the purpose of routing is to find a way to get datagrams to their ultimate destinations. Distance vector algorithms are based on a table giving the best route to every destination in the system. Of course, in order to define which route is best, we have to have some way of measuring goodness. This is referred to as the "metric".

In simple networks, it is common to use a metric that simply counts how many gateways a message must go through. In more complex networks, a metric is chosen to represent the total amount of delay that the message suffers, the cost of sending it, or some other quantity which may be minimized. The main requirement is that it must be possible to represent the metric as a sum of "costs" for individual hops.

Formally, if it is possible to get from entity i to entity j directly (i.e., without passing through another gateway between), then a cost, d(i,j), is associated with the hop between i and j. In the normal case where all entities on a given network are considered to be the same, d(i,j) is the same for all destinations on a given network, and represents the cost of using that network. To get the metric of a complete route, one just adds up the costs of the individual hops that make up the route. For the purposes of this memo, we assume that the costs are positive integers.

Let D(i,j) represent the metric of the best route from entity i to entity j. It should be defined for every pair of entities. d(i,j) represents the costs of the individual steps. Formally, let d(i,j) represent the cost of going directly from entity i to entity j. It is infinite if i and j are not immediate neighbors. (Note that d(i,i)

is infinite. That is, we don't consider there to be a direct connection from a node to itself.) Since costs are additive, it is easy to show that the best metric must be described by

$$D(i,i) = 0, \qquad \text{all } i$$
$$D(i,j) = \min_k [d(i,k) + D(k,j)], \qquad \text{otherwise}$$

and that the best routes start by going from i to those neighbors k for which d(i,k) + D(k,j) has the minimum value. (These things can be shown by induction on the number of steps in the routes.) Note that we can limit the second equation to k's that are immediate neighbors of i. For the others, d(i,k) is infinite, so the term involving them can never be the minimum.

It turns out that one can compute the metric by a simple algorithm based on this. Entity i gets its neighbors k to send it their estimates of their distances to the destination j. When i gets the estimates from k, it adds d(i,k) to each of the numbers. This is simply the cost of traversing the network between i and k. Now and then i compares the values from all of its neighbors and picks the smallest.

A proof is given in [2] that this algorithm will converge to the correct estimates of D(i,j) in finite time in the absence of topology changes. The authors make very few assumptions about the order in which the entities send each other their information, or when the min is recomputed. Basically, entities just can't stop sending updates or recomputing metrics, and the networks can't delay messages forever. (Crash of a routing entity is a topology change.) Also, their proof does not make any assumptions about the initial estimates of D(i,j), except that they must be non-negative. The fact that these fairly weak assumptions are good enough is important. Because we don't have to make assumptions about when updates are sent, it is safe to run the algorithm asynchronously. That is, each entity can send updates according to its own clock. Updates can be dropped by the network, as long as they don't all get dropped. Because we don't have to make assumptions about the starting condition, the algorithm can handle changes. When the system changes, the routing algorithm starts moving to a new equilibrium, using the old one as its starting point. It is important that the algorithm will converge in finite time no matter what the starting point. Otherwise certain kinds of changes might lead to non-convergent behavior.

The statement of the algorithm given above (and the proof) assumes that each entity keeps copies of the estimates that come from each of its neighbors, and now and then does a min over all of the neighbors. In fact real implementations don't necessarily do that. They simply

remember the best metric seen so far, and the identity of the neighbor that sent it. They replace this information whenever they see a better (smaller) metric. This allows them to compute the minimum incrementally, without having to store data from all of the neighbors.

There is one other difference between the algorithm as described in texts and those used in real protocols such as RIP: the description above would have each entity include an entry for itself, showing a distance of zero. In fact this is not generally done. Recall that all entities on a network are normally summarized by a single entry for the network. Consider the situation of a host or gateway G that is connected to network A. C represents the cost of using network A (usually a metric of one). (Recall that we are assuming that the internal structure of a network is not visible to IP, and thus the cost of going between any two entities on it is the same.) In principle, G should get a message from every other entity H on network A, showing a cost of 0 to get from that entity to itself. G would then compute C + 0 as the distance to H. Rather than having G look at all of these identical messages, it simply starts out by making an entry for network A in its table, and assigning it a metric of C. This entry for network A should be thought of as summarizing the entries for all other entities on network A. The only entity on A that can't be summarized by that common entry is G itself, since the cost of going from G to G is 0, not C. But since we never need those 0 entries, we can safely get along with just the single entry for network A. Note one other implication of this strategy: because we don't need to use the 0 entries for anything, hosts that do not function as gateways don't need to send any update messages. Clearly hosts that don't function as gateways (i.e., hosts that are connected to only one network) can have no useful information to contribute other than their own entry D(i,i) = 0. As they have only the one interface, it is easy to see that a route to any other network through them will simply go in that interface and then come right back out it. Thus the cost of such a route will be greater than the best cost by at least C. Since we don't need the 0 entries, non-gateways need not participate in the routing protocol at all.

Let us summarize what a host or gateway G does. For each destination in the system, G will keep a current estimate of the metric for that destination (i.e., the total cost of getting to it) and the identity of the neighboring gateway on whose data that metric is based. If the destination is on a network that is directly connected to G, then G simply uses an entry that shows the cost of using the network, and the fact that no gateway is needed to get to the destination. It is easy to show that once the computation has converged to the correct metrics, the neighbor that is recorded by this technique is in fact the first gateway on the path to the destination. (If there are

several equally good paths, it is the first gateway on one of them.) This combination of destination, metric, and gateway is typically referred to as a route to the destination with that metric, using that gateway.

The method so far only has a way to lower the metric, as the existing metric is kept until a smaller one shows up. It is possible that the initial estimate might be too low. Thus, there must be a way to increase the metric. It turns out to be sufficient to use the following rule: suppose the current route to a destination has metric D and uses gateway G. If a new set of information arrived from some source other than G, only update the route if the new metric is better than D. But if a new set of information arrives from G itself, always update D to the new value. It is easy to show that with this rule, the incremental update process produces the same routes as a calculation that remembers the latest information from all the neighbors and does an explicit minimum. (Note that the discussion so far assumes that the network configuration is static. It does not allow for the possibility that a system might fail.)

To summarize, here is the basic distance vector algorithm as it has been developed so far. (Note that this is not a statement of the RIP protocol. There are several refinements still to be added.) The following procedure is carried out by every entity that participates in the routing protocol. This must include all of the gateways in the system. Hosts that are not gateways may participate as well.

- Keep a table with an entry for every possible destination in the system. The entry contains the distance D to the destination, and the first gateway G on the route to that network. Conceptually, there should be an entry for the entity itself, with metric 0, but this is not actually included.

- Periodically, send a routing update to every neighbor. The update is a set of messages that contain all of the information from the routing table. It contains an entry for each destination, with the distance shown to that destination.

- When a routing update arrives from a neighbor G', add the cost associated with the network that is shared with G'. (This should be the network over which the update arrived.) Call the resulting distance D'. Compare the resulting distances with the current routing table entries. If the new distance D' for N is smaller than the existing value D, adopt the new route. That is, change the table entry for N to have metric D' and gateway G'. If G' is the gateway

from which the existing route came, i.e., G' = G, then use the new metric even if it is larger than the old one.

2.1. Dealing with changes in topology

The discussion above assumes that the topology of the network is fixed. In practice, gateways and lines often fail and come back up. To handle this possibility, we need to modify the algorithm slightly. The theoretical version of the algorithm involved a minimum over all immediate neighbors. If the topology changes, the set of neighbors changes. Therefore, the next time the calculation is done, the change will be reflected. However, as mentioned above, actual implementations use an incremental version of the minimization. Only the best route to any given destination is remembered. If the gateway involved in that route should crash, or the network connection to it break, the calculation might never reflect the change. The algorithm as shown so far depends upon a gateway notifying its neighbors if its metrics change. If the gateway crashes, then it has no way of notifying neighbors of a change.

In order to handle problems of this kind, distance vector protocols must make some provision for timing out routes. The details depend upon the specific protocol. As an example, in RIP every gateway that participates in routing sends an update message to all its neighbors once every 30 seconds. Suppose the current route for network N uses gateway G. If we don't hear from G for 180 seconds, we can assume that either the gateway has crashed or the network connecting us to it has become unusable. Thus, we mark the route as invalid. When we hear from another neighbor that has a valid route to N, the valid route will replace the invalid one. Note that we wait for 180 seconds before timing out a route even though we expect to hear from each neighbor every 30 seconds. Unfortunately, messages are occasionally lost by networks. Thus, it is probably not a good idea to invalidate a route based on a single missed message.

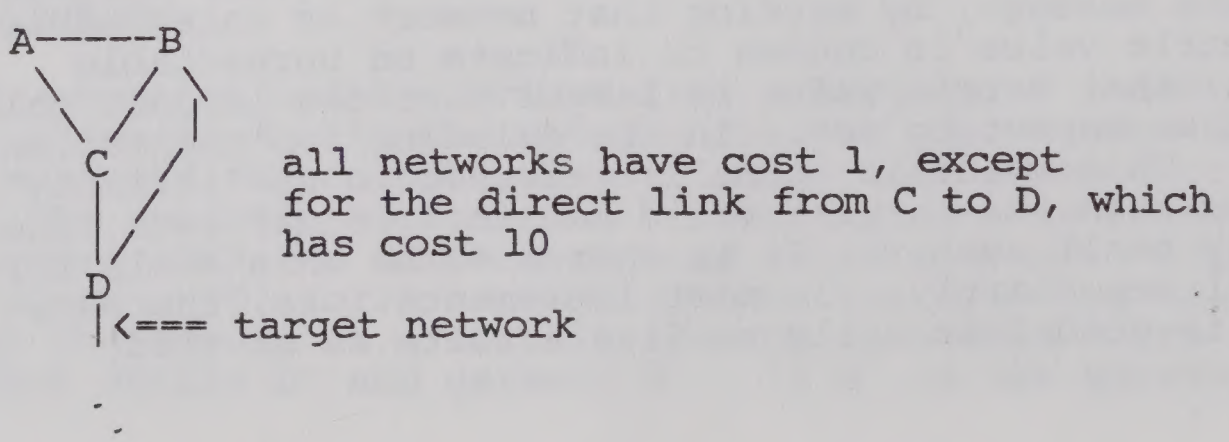
As we will see below, it is useful to have a way to notify neighbors that there currently isn't a valid route to some network. RIP, along with several other protocols of this class, does this through a normal update message, by marking that network as unreachable. A specific metric value is chosen to indicate an unreachable destination; that metric value is larger than the largest valid metric that we expect to see. In the existing implementation of RIP, 16 is used. This value is normally referred to as "infinity", since it is larger than the largest valid metric. 16 may look like a surprisingly small number. It is chosen to be this small for reasons that we will see shortly. In most implementations, the same convention is used internally to flag a route as invalid.

2.2. Preventing instability

The algorithm as presented up to this point will always allow a host or gateway to calculate a correct routing table. However, that is still not quite enough to make it useful in practice. The proofs referred to above only show that the routing tables will converge to the correct values in finite time. They do not guarantee that this time will be small enough to be useful, nor do they say what will happen to the metrics for networks that become inaccessible.

It is easy enough to extend the mathematics to handle routes becoming inaccessible. The convention suggested above will do that. We choose a large metric value to represent "infinity". This value must be large enough that no real metric would ever get that large. For the purposes of this example, we will use the value 16. Suppose a network becomes inaccessible. All of the immediately neighboring gateways time out and set the metric for that network to 16. For purposes of analysis, we can assume that all the neighboring gateways have gotten a new piece of hardware that connects them directly to the vanished network, with a cost of 16. Since that is the only connection to the vanished network, all the other gateways in the system will converge to new routes that go through one of those gateways. It is easy to see that once convergence has happened, all the gateways will have metrics of at least 16 for the vanished network. Gateways one hop away from the original neighbors would end up with metrics of at least 17; gateways two hops away would end up with at least 18, etc. As these metrics are larger than the maximum metric value, they are all set to 16. It is obvious that the system will now converge to a metric of 16 for the vanished network at all gateways.

Unfortunately, the question of how long convergence will take is not amenable to quite so simple an answer. Before going any further, it will be useful to look at an example (taken from [2]). Note, by the way, that what we are about to show will not happen with a correct implementation of RIP. We are trying to show why certain features are needed. Note that the letters correspond to gateways, and the lines to networks.



Each gateway will have a table showing a route to each network.

However, for purposes of this illustration, we show only the routes from each gateway to the network marked at the bottom of the diagram.

D: directly connected, metric 1
B: route via D, metric 2
C: route via B, metric 3
A: route via B, metric 3

Now suppose that the link from B to D fails. The routes should now adjust to use the link from C to D. Unfortunately, it will take a while for this to this to happen. The routing changes start when B notices that the route to D is no longer usable. For simplicity, the chart below assumes that all gateways send updates at the same time. The chart shows the metric for the target network, as it appears in the routing table at each gateway.

```
time ------>

D: dir, 1   dir, 1   dir, 1   dir, 1  ...  dir, 1   dir, 1
B: unreach  C,   4   C,   5   C,   6       C,  11   C,  12
C: B,   3   A,   4   A,   5   A,   6       A,  11   D,  11
A: B,   3   C,   4   C,   5   C,   6       C,  11   C,  12

dir = directly connected
unreach = unreachable
```

Here's the problem: B is able to get rid of its failed route using a timeout mechanism. But vestiges of that route persist in the system for a long time. Initially, A and C still think they can get to D via B. So, they keep sending updates listing metrics of 3. In the next iteration, B will then claim that it can get to D via either A or C. Of course, it can't. The routes being claimed by A and C are now gone, but they have no way of knowing that yet. And even when they discover that their routes via B have gone away, they each think there is a route available via the other. Eventually the system converges, as all the mathematics claims it must. But it can take some time to do so. The worst case is when a network becomes completely inaccessible from some part of the system. In that case, the metrics may increase slowly in a pattern like the one above until they finally reach infinity. For this reason, the problem is called "counting to infinity".

You should now see why "infinity" is chosen to be as small as possible. If a network becomes completely inaccessible, we want counting to infinity to be stopped as soon as possible. Infinity must be large enough that no real route is that big. But it

shouldn't be any bigger than required. Thus the choice of infinity is a tradeoff between network size and speed of convergence in case counting to infinity happens. The designers of RIP believed that the protocol was unlikely to be practical for networks with a diameter larger than 15.

There are several things that can be done to prevent problems like this. The ones used by RIP are called "split horizon with poisoned reverse", and "triggered updates".

2.2.1. Split horizon

Note that some of the problem above is caused by the fact that A and C are engaged in a pattern of mutual deception. Each claims to be able to get to D via the other. This can be prevented by being a bit more careful about where information is sent. In particular, it is never useful to claim reachability for a destination network to the neighbor(s) from which the route was learned. "Split horizon" is a scheme for avoiding problems caused by including routes in updates sent to the gateway from which they were learned. The "simple split horizon" scheme omits routes learned from one neighbor in updates sent to that neighbor. "Split horizon with poisoned reverse" includes such routes in updates, but sets their metrics to infinity.

If A thinks it can get to D via C, its messages to C should indicate that D is unreachable. If the route through C is real, then C either has a direct connection to D, or a connection through some other gateway. C's route can't possibly go back to A, since that forms a loop. By telling C that D is unreachable, A simply guards against the possibility that C might get confused and believe that there is a route through A. This is obvious for a point to point line. But consider the possibility that A and C are connected by a broadcast network such as an Ethernet, and there are other gateways on that network. If A has a route through C, it should indicate that D is unreachable when talking to any other gateway on that network. The other gateways on the network can get to C themselves. They would never need to get to C via A. If A's best route is really through C, no other gateway on that network needs to know that A can reach D. This is fortunate, because it means that the same update message that is used for C can be used for all other gateways on the same network. Thus, update messages can be sent by broadcast.

In general, split horizon with poisoned reverse is safer than simple split horizon. If two gateways have routes pointing at each other, advertising reverse routes with a metric of 16 will break the loop immediately. If the reverse routes are simply not advertised, the erroneous routes will have to be eliminated by waiting for a timeout. However, poisoned reverse does have a disadvantage: it increases the

size of the routing messages. Consider the case of a campus backbone connecting a number of different buildings. In each building, there is a gateway connecting the backbone to a local network. Consider what routing updates those gateways should broadcast on the backbone network. All that the rest of the network really needs to know about each gateway is what local networks it is connected to. Using simple split horizon, only those routes would appear in update messages sent by the gateway to the backbone network. If split horizon with poisoned reverse is used, the gateway must mention all routes that it learns from the backbone, with metrics of 16. If the system is large, this can result in a large update message, almost all of whose entries indicate unreachable networks.

In a static sense, advertising reverse routes with a metric of 16 provides no additional information. If there are many gateways on one broadcast network, these extra entries can use significant bandwidth. The reason they are there is to improve dynamic behavior. When topology changes, mentioning routes that should not go through the gateway as well as those that should can speed up convergence. However, in some situations, network managers may prefer to accept somewhat slower convergence in order to minimize routing overhead. Thus implementors may at their option implement simple split horizon rather than split horizon with poisoned reverse, or they may provide a configuration option that allows the network manager to choose which behavior to use. It is also permissible to implement hybrid schemes that advertise some reverse routes with a metric of 16 and omit others. An example of such a scheme would be to use a metric of 16 for reverse routes for a certain period of time after routing changes involving them, and thereafter omitting them from updates.

2.2.2. Triggered updates

Split horizon with poisoned reverse will prevent any routing loops that involve only two gateways. However, it is still possible to end up with patterns in which three gateways are engaged in mutual deception. For example, A may believe it has a route through B, B through C, and C through A. Split horizon cannot stop such a loop. This loop will only be resolved when the metric reaches infinity and the network involved is then declared unreachable. Triggered updates are an attempt to speed up this convergence. To get triggered updates, we simply add a rule that whenever a gateway changes the metric for a route, it is required to send update messages almost immediately, even if it is not yet time for one of the regular update message. (The timing details will differ from protocol to protocol. Some distance vector protocols, including RIP, specify a small time delay, in order to avoid having triggered updates generate excessive network traffic.) Note how this combines with the rules for computing new metrics. Suppose a gateway's route to destination N

goes through gateway G. If an update arrives from G itself, the receiving gateway is required to believe the new information, whether the new metric is higher or lower than the old one. If the result is a change in metric, then the receiving gateway will send triggered updates to all the hosts and gateways directly connected to it. They in turn may each send updates to their neighbors. The result is a cascade of triggered updates. It is easy to show which gateways and hosts are involved in the cascade. Suppose a gateway G times out a route to destination N. G will send triggered updates to all of its neighbors. However, the only neighbors who will believe the new information are those whose routes for N go through G. The other gateways and hosts will see this as information about a new route that is worse than the one they are already using, and ignore it. The neighbors whose routes go through G will update their metrics and send triggered updates to all of their neighbors. Again, only those neighbors whose routes go through them will pay attention. Thus, the triggered updates will propagate backwards along all paths leading to gateway G, updating the metrics to infinity. This propagation will stop as soon as it reaches a portion of the network whose route to destination N takes some other path.

If the system could be made to sit still while the cascade of triggered updates happens, it would be possible to prove that counting to infinity will never happen. Bad routes would always be removed immediately, and so no routing loops could form.

Unfortunately, things are not so nice. While the triggered updates are being sent, regular updates may be happening at the same time. Gateways that haven't received the triggered update yet will still be sending out information based on the route that no longer exists. It is possible that after the triggered update has gone through a gateway, it might receive a normal update from one of these gateways that hasn't yet gotten the word. This could reestablish an orphaned remnant of the faulty route. If triggered updates happen quickly enough, this is very unlikely. However, counting to infinity is still possible.

3. Specifications for the protocol

RIP is intended to allow hosts and gateways to exchange information for computing routes through an IP-based network. RIP is a distance vector protocol. Thus, it has the general features described in section 2. RIP may be implemented by both hosts and gateways. As in most IP documentation, the term "host" will be used here to cover either. RIP is used to convey information about routes to "destinations", which may be individual hosts, networks, or a special destination used to convey a default route.

Any host that uses RIP is assumed to have interfaces to one or more networks. These are referred to as its "directly-connected networks". The protocol relies on access to certain information about each of these networks. The most important is its metric or "cost". The metric of a network is an integer between 1 and 15 inclusive. It is set in some manner not specified in this protocol. Most existing implementations always use a metric of 1. New implementations should allow the system administrator to set the cost of each network. In addition to the cost, each network will have an IP network number and a subnet mask associated with it. These are to be set by the system administrator in a manner not specified in this protocol.

Note that the rules specified in section 3.2 assume that there is a single subnet mask applying to each IP network, and that only the subnet masks for directly-connected networks are known. There may be systems that use different subnet masks for different subnets within a single network. There may also be instances where it is desirable for a system to know the subnets masks of distant networks. However, such situations will require modifications of the rules which govern the spread of subnet information. Such modifications raise issues of interoperability, and thus must be viewed as modifying the protocol.

Each host that implements RIP is assumed to have a routing table. This table has one entry for every destination that is reachable through the system described by RIP. Each entry contains at least the following information:

- The IP address of the destination.

- A metric, which represents the total cost of getting a datagram from the host to that destination. This metric is the sum of the costs associated with the networks that would be traversed in getting to the destination.

- The IP address of the next gateway along the path to the destination. If the destination is on one of the directly-connected networks, this item is not needed.

- A flag to indicate that information about the route has changed recently. This will be referred to as the "route change flag."

- Various timers associated with the route. See section 3.3 for more details on them.

The entries for the directly-connected networks are set up by the host, using information gathered by means not specified in this

protocol. The metric for a directly-connected network is set to the cost of that network. In existing RIP implementations, 1 is always used for the cost. In that case, the RIP metric reduces to a simple hop-count. More complex metrics may be used when it is desirable to show preference for some networks over others, for example because of differences in bandwidth or reliability.

Implementors may also choose to allow the system administrator to enter additional routes. These would most likely be routes to hosts or networks outside the scope of the routing system.

Entries for destinations other these initial ones are added and updated by the algorithms described in the following sections.

In order for the protocol to provide complete information on routing, every gateway in the system must participate in it. Hosts that are not gateways need not participate, but many implementations make provisions for them to listen to routing information in order to allow them to maintain their routing tables.

3.1. Message formats

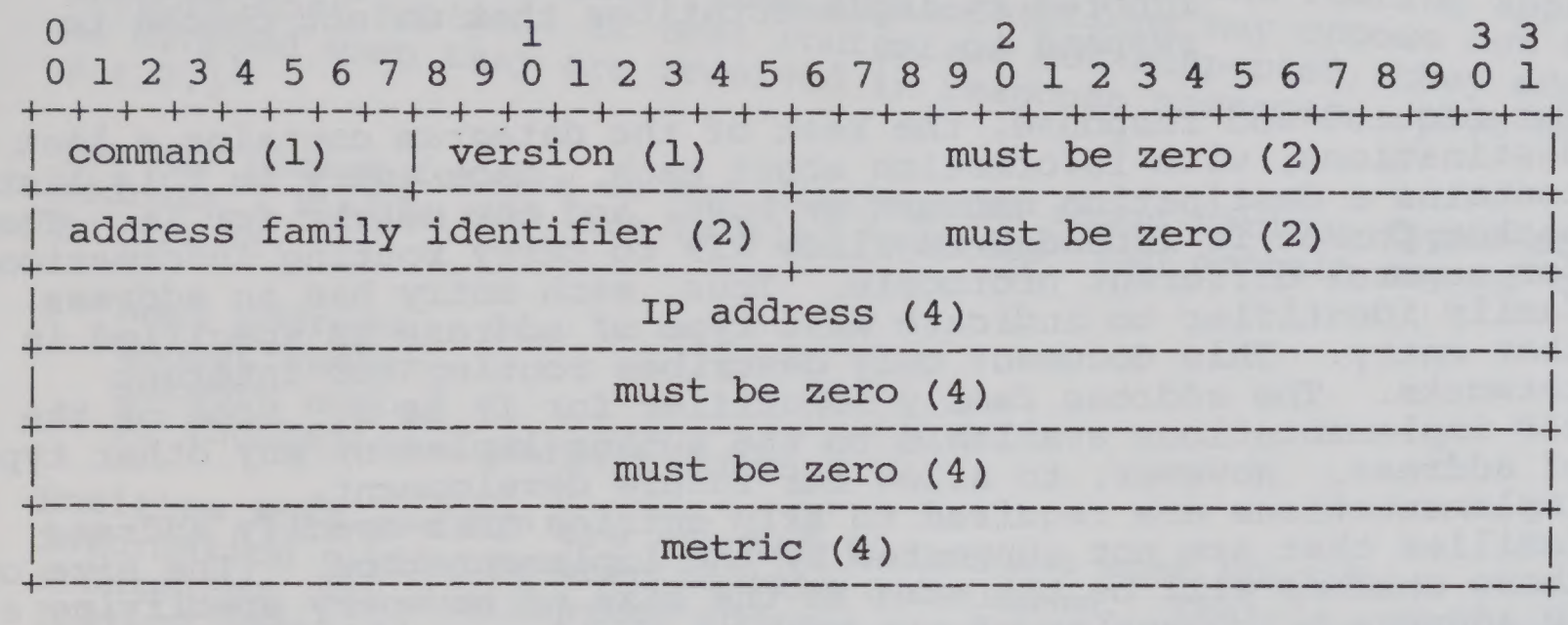
RIP is a UDP-based protocol. Each host that uses RIP has a routing process that sends and receives datagrams on UDP port number 520. All communications directed at another host's RIP processor are sent to port 520. All routing update messages are sent from port 520. Unsolicited routing update messages have both the source and destination port equal to 520. Those sent in response to a request are sent to the port from which the request came. Specific queries and debugging requests may be sent from ports other than 520, but they are directed to port 520 on the target machine.

There are provisions in the protocol to allow "silent" RIP processes. A silent process is one that normally does not send out any messages. However, it listens to messages sent by others. A silent RIP might be used by hosts that do not act as gateways, but wish to listen to routing updates in order to monitor local gateways and to keep their internal routing tables up to date. (See [5] for a discussion of various ways that hosts can keep track of network topology.) A gateway that has lost contact with all but one of its networks might choose to become silent, since it is effectively no longer a gateway.

However, this should not be done if there is any chance that neighboring gateways might depend upon its messages to detect that the failed network has come back into operation. (The 4BSD routed program uses routing packets to monitor the operation of point-to-point links.)

The packet format is shown in Figure 1.

Format of datagrams containing network information. Field sizes are given in octets. Unless otherwise specified, fields contain binary integers, in normal Internet order with the most-significant octet first. Each tick mark represents one bit.


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
command (1)
version (1)
must be zero (2)
address family identifier (2)
must be zero (2)
IP address (4)
must be zero (4)
must be zero (4)
metric (4)

The portion of the datagram from address family identifier through metric may appear up to 25 times. IP address is the usual 4-octet Internet address, in network order.

Figure 1. Packet format

Every datagram contains a command, a version number, and possible arguments. This document describes version 1 of the protocol. Details of processing the version number are described in section 3.4. The command field is used to specify the purpose of this datagram. Here is a summary of the commands implemented in version 1:

1 - request	A request for the responding system to send all or part of its routing table.
2 - response	A message containing all or part of the sender's routing table. This message may be sent in response to a request or poll, or it may be an update message generated by the sender.
3 - traceon	Obsolete. Messages containing this command are to be ignored.

4 - traceoff Obsolete. Messages containing this command are to be ignored.

5 - reserved This value is used by Sun Microsystems for its own purposes. If new commands are added in any succeeding version, they should begin with 6. Messages containing this command may safely be ignored by implementations that do not choose to respond to it.

For request and response, the rest of the datagram contains a list of destinations, with information about each. Each entry in this list contains a destination network or host, and the metric for it. The packet format is intended to allow RIP to carry routing information for several different protocols. Thus, each entry has an address family identifier to indicate what type of address is specified in that entry. This document only describes routing for Internet networks. The address family identifier for IP is 2. None of the RIP implementations available to the author implement any other type of address. However, to allow for future development, implementations are required to skip entries that specify address families that are not supported by the implementation. (The size of these entries will be the same as the size of an entry specifying an IP address.) Processing of the message continues normally after any unsupported entries are skipped. The IP address is the usual Internet address, stored as 4 octets in network order. The metric field must contain a value between 1 and 15 inclusive, specifying the current metric for the destination, or the value 16, which indicates that the destination is not reachable. Each route sent by a gateway supercedes any previous route to the same destination from the same gateway.

The maximum datagram size is 512 octets. This includes only the portions of the datagram described above. It does not count the IP or UDP headers. The commands that involve network information allow information to be split across several datagrams. No special provisions are needed for continuations, since correct results will occur if the datagrams are processed individually.

3.2. Addressing considerations

As indicated in section 2, distance vector routing can be used to describe routes to individual hosts or to networks. The RIP protocol allows either of these possibilities. The destinations appearing in request and response messages can be networks, hosts, or a special code used to indicate a default address. In general, the kinds of routes actually used will depend upon the routing strategy used for the particular network. Many networks are set up so that routing

information for individual hosts is not needed. If every host on a given network or subnet is accessible through the same gateways, then there is no reason to mention individual hosts in the routing tables. However, networks that include point to point lines sometimes require gateways to keep track of routes to certain hosts. Whether this feature is required depends upon the addressing and routing approach used in the system. Thus, some implementations may choose not to support host routes. If host routes are not supported, they are to be dropped when they are received in response messages. (See section 3.4.2.)

The RIP packet formats do not distinguish among various types of address. Fields that are labeled "address" can contain any of the following:

host address
subnet number
network number
0, indicating a default route

Entities that use RIP are assumed to use the most specific information available when routing a datagram. That is, when routing a datagram, its destination address must first be checked against the list of host addresses. Then it must be checked to see whether it matches any known subnet or network number. Finally, if none of these match, the default route is used.

When a host evaluates information that it receives via RIP, its interpretation of an address depends upon whether it knows the subnet mask that applies to the net. If so, then it is possible to determine the meaning of the address. For example, consider net 128.6. It has a subnet mask of 255.255.255.0. Thus 128.6.0.0 is a network number, 128.6.4.0 is a subnet number, and 128.6.4.1 is a host address. However, if the host does not know the subnet mask, evaluation of an address may be ambiguous. If there is a non-zero host part, there is no clear way to determine whether the address represents a subnet number or a host address. As a subnet number would be used without the subnet mask, addresses are assumed to represent hosts in this situation. In order to avoid this sort of ambiguity, hosts must not send subnet routes to hosts that cannot be expected to know the appropriate subnet mask. Normally hosts only know the subnet masks for directly-connected networks. Therefore, unless special provisions have been made, routes to a subnet must not be sent outside the network of which the subnet is a part.

This filtering is carried out by the gateways at the "border" of the subnetted network. These are gateways that connect that network with some other network. Within the subnetted network, each subnet is

treated as an individual network. Routing entries for each subnet are circulated by RIP. However, border gateways send only a single entry for the network as a whole to hosts in other networks. This means that a border gateway will send different information to different neighbors. For neighbors connected to the subnetted network, it generates a list of all subnets to which it is directly connected, using the subnet number. For neighbors connected to other networks, it makes a single entry for the network as a whole, showing the metric associated with that network. (This metric would normally be the smallest metric for the subnets to which the gateway is attached.)

Similarly, border gateways must not mention host routes for hosts within one of the directly-connected networks in messages to other networks. Those routes will be subsumed by the single entry for the network as a whole. We do not specify what to do with host routes for "distant" hosts (i.e., hosts not part of one of the directly-connected networks). Generally, these routes indicate some host that is reachable via a route that does not support other hosts on the network of which the host is a part.

The special address 0.0.0.0 is used to describe a default route. A default route is used when it is not convenient to list every possible network in the RIP updates, and when one or more closely-connected gateways in the system are prepared to handle traffic to the networks that are not listed explicitly. These gateways should create RIP entries for the address 0.0.0.0, just as if it were a network to which they are connected. The decision as to how gateways create entries for 0.0.0.0 is left to the implementor. Most commonly, the system administrator will be provided with a way to specify which gateways should create entries for 0.0.0.0. However, other mechanisms are possible. For example, an implementor might decide that any gateway that speaks EGP should be declared to be a default gateway. It may be useful to allow the network administrator to choose the metric to be used in these entries. If there is more than one default gateway, this will make it possible to express a preference for one over the other. The entries for 0.0.0.0 are handled by RIP in exactly the same manner as if there were an actual network with this address. However, the entry is used to route any datagram whose destination address does not match any other network in the table. Implementations are not required to support this convention. However, it is strongly recommended. Implementations that do not support 0.0.0.0 must ignore entries with this address. In such cases, they must not pass the entry on in their own RIP updates. System administrators should take care to make sure that routes to 0.0.0.0 do not propagate further than is intended. Generally, each autonomous system has its own preferred default gateway. Thus, routes involving 0.0.0.0 should generally not leave

the boundary of an autonomous system. The mechanisms for enforcing this are not specified in this document.

3.3. Timers

This section describes all events that are triggered by timers.

Every 30 seconds, the output process is instructed to generate a complete response to every neighboring gateway. When there are many gateways on a single network, there is a tendency for them to synchronize with each other such that they all issue updates at the same time. This can happen whenever the 30 second timer is affected by the processing load on the system. It is undesirable for the update messages to become synchronized, since it can lead to unnecessary collisions on broadcast networks. Thus, implementations are required to take one of two precautions.

- The 30-second updates are triggered by a clock whose rate is not affected by system load or the time required to service the previous update timer.

- The 30-second timer is offset by addition of a small random time each time it is set.

There are two timers associated with each route, a "timeout" and a "garbage-collection time". Upon expiration of the timeout, the route is no longer valid. However, it is retained in the table for a short time, so that neighbors can be notified that the route has been dropped. Upon expiration of the garbage-collection timer, the route is finally removed from the tables.

The timeout is initialized when a route is established, and any time an update message is received for the route. If 180 seconds elapse from the last time the timeout was initialized, the route is considered to have expired, and the deletion process which we are about to describe is started for it.

Deletions can occur for one of two reasons: (1) the timeout expires, or (2) the metric is set to 16 because of an update received from the current gateway. (See section 3.4.2 for a discussion processing updates from other gateways.) In either case, the following events happen:

- The garbage-collection timer is set for 120 seconds.

- The metric for the route is set to 16 (infinity). This causes the route to be removed from service.

- A flag is set noting that this entry has been changed, and the output process is signalled to trigger a response.

Until the garbage-collection timer expires, the route is included in all updates sent by this host, with a metric of 16 (infinity). When the garbage-collection timer expires, the route is deleted from the tables.

Should a new route to this network be established while the garbage-collection timer is running, the new route will replace the one that is about to be deleted. In this case the garbage-collection timer must be cleared.

See section 3.5 for a discussion of a delay that is required in carrying out triggered updates. Although implementation of that delay will require a timer, it is more natural to discuss it in section 3.5 than here.

3.4. Input processing

This section will describe the handling of datagrams received on UDP port 520. Before processing the datagrams in detail, certain general format checks must be made. These depend upon the version number field in the datagram, as follows:

0 Datagrams whose version number is zero are to be ignored. These are from a previous version of the protocol, whose packet format was machine-specific.

1 Datagrams whose version number is one are to be processed as described in the rest of this specification. All fields that are described above as "must be zero" are to be checked. If any such field contains a non-zero value, the entire message is to be ignored.

>1 Datagrams whose version number are greater than one are to be processed as described in the rest of this specification. All fields that are described above as "must be zero" are to be ignored. Future versions of the protocol may put data into these fields. Version 1 implementations are to ignore this extra data and process only the fields specified in this document.

After checking the version number and doing any other preliminary checks, processing will depend upon the value in the command field.

3.4.1. Request

Request is used to ask for a response containing all or part of the host's routing table. [Note that the term host is used for either host or gateway, in most cases it would be unusual for a non-gateway host to send RIP messages.] Normally, requests are sent as broadcasts, from a UDP source port of 520. In this case, silent processes do not respond to the request. Silent processes are by definition processes for which we normally do not want to see routing information. However, there may be situations involving gateway monitoring where it is desired to look at the routing table even for a silent process. In this case, the request should be sent from a UDP port number other than 520. If a request comes from port 520, silent processes do not respond. If the request comes from any other port, processes must respond even if they are silent.

The request is processed entry by entry. If there are no entries, no response is given. There is one special case. If there is exactly one entry in the request, with an address family identifier of 0 (meaning unspecified), and a metric of infinity (i.e., 16 for current implementations), this is a request to send the entire routing table. In that case, a call is made to the output process to send the routing table to the requesting port.

Except for this special case, processing is quite simple. Go down the list of entries in the request one by one. For each entry, look up the destination in the host's routing database. If there is a route, put that route's metric in the metric field in the datagram. If there isn't a route to the specified destination, put infinity (i.e., 16) in the metric field in the datagram. Once all the entries have been filled in, set the command to response and send the datagram back to the port from which it came.

Note that there is a difference in handling depending upon whether the request is for a specified set of destinations, or for a complete routing table. If the request is for a complete host table, normal output processing is done. This includes split horizon (see section 2.2.1) and subnet hiding (section 3.2), so that certain entries from the routing table will not be shown. If the request is for specific entries, they are looked up in the host table and the information is returned. No split horizon processing is done, and subnets are returned if requested. We anticipate that these requests are likely to be used for different purposes. When a host first comes up, it broadcasts requests on every connected network asking for a complete routing table. In general, we assume that complete routing tables are likely to be used to update another host's routing table. For this reason, split horizon and all other filtering must be used. Requests for specific networks are made only by diagnostic software,

and are not used for routing. In this case, the requester would want to know the exact contents of the routing database, and would not want any information hidden.

3.4.2. Response

Responses can be received for several different reasons:

response to a specific query
regular updates
triggered updates triggered by a metric change

Processing is the same no matter how responses were generated.

Because processing of a response may update the host's routing table, the response must be checked carefully for validity. The response must be ignored if it is not from port 520. The IP source address should be checked to see whether the datagram is from a valid neighbor. The source of the datagram must be on a directly-connected network. It is also worth checking to see whether the response is from one of the host's own addresses. Interfaces on broadcast networks may receive copies of their own broadcasts immediately. If a host processes its own output as new input, confusion is likely, and such datagrams must be ignored (except as discussed in the next paragraph).

Before actually processing a response, it may be useful to use its presence as input to a process for keeping track of interface status. As mentioned above, we time out a route when we haven't heard from its gateway for a certain amount of time. This works fine for routes that come from another gateway. It is also desirable to know when one of our own directly-connected networks has failed. This document does not specify any particular method for doing this, as such methods depend upon the characteristics of the network and the hardware interface to it. However, such methods often involve listening for datagrams arriving on the interface. Arriving datagrams can be used as an indication that the interface is working. However, some caution must be used, as it is possible for interfaces to fail in such a way that input datagrams are received, but output datagrams are never sent successfully.

Now that the datagram as a whole has been validated, process the entries in it one by one. Again, start by doing validation. If the metric is greater than infinity, ignore the entry. (This should be impossible, if the other host is working correctly. Incorrect metrics and other format errors should probably cause alerts or be logged.) Then look at the destination address. Check the address family identifier. If it is not a value which is expected (e.g., 2

for Internet addresses), ignore the entry. Now check the address itself for various kinds of inappropriate addresses. Ignore the entry if the address is class D or E, if it is on net 0 (except for 0.0.0.0, if we accept default routes) or if it is on net 127 (the loopback network). Also, test for a broadcast address, i.e., anything whose host part is all ones on a network that supports broadcast, and ignore any such entry. If the implementor has chosen not to support host routes (see section 3.2), check to see whether the host portion of the address is non-zero; if so, ignore the entry.

Recall that the address field contains a number of unused octets. If the version number of the datagram is 1, they must also be checked. If any of them is nonzero, the entry is to be ignored. (Many of these cases indicate that the host from which the message came is not working correctly. Thus some form of error logging or alert should be triggered.)

Update the metric by adding the cost of the network on which the message arrived. If the result is greater than 16, use 16. That is,

```
metric = MIN (metric + cost, 16)
```

Now look up the address to see whether this is already a route for it. In general, if not, we want to add one. However, there are various exceptions. If the metric is infinite, don't add an entry. (We would update an existing one, but we don't add new entries with infinite metric.) We want to avoid adding routes to hosts if the host is part of a net or subnet for which we have at least as good a route. If neither of these exceptions applies, add a new entry to the routing database. This includes the following actions:

- Set the destination and metric to those from the datagram.
- Set the gateway to be the host from which the datagram came.
- Initialize the timeout for the route. If the garbage-collection timer is running for this route, stop it. (See section 3.3 for a discussion of the timers.)
- Set the route change flag, and signal the output process to trigger an update (see 3.5).

If there is an existing route, first compare gateways. If this datagram is from the same gateway as the existing route, reinitialize the timeout. Next compare metrics. If the datagram is from the same gateway as the existing route and the new metric is different than the old one, or if the new metric is lower than the old one, do the

following actions:

- adopt the route from the datagram. That is, put the new metric in, and set the gateway to be the host from which the datagram came.

- Initialize the timeout for the route.

- Set the route change flag, and signal the output process to trigger an update (see 3.5).

- If the new metric is 16 (infinity), the deletion process is started.

If the new metric is 16 (infinity), this starts the process for deleting the route. The route is no longer used for routing packets, and the deletion timer is started (see section 3.3). Note that a deletion is started only when the metric is first set to 16. If the metric was already 16, then a new deletion is not started. (Starting a deletion sets a timer. The concern is that we do not want to reset the timer every 30 seconds, as new messages arrive with an infinite metric.)

If the new metric is the same as the old one, it is simplest to do nothing further (beyond reinitializing the timeout, as specified above). However, the 4BSD routed uses an additional heuristic here. Normally, it is senseless to change to a route with the same metric as the existing route but a different gateway. If the existing route is showing signs of timing out, though, it may be better to switch to an equally-good alternative route immediately, rather than waiting for the timeout to happen. (See section 3.3 for a discussion of timeouts.) Therefore, if the new metric is the same as the old one, routed looks at the timeout for the existing route. If it is at least halfway to the expiration point, routed switches to the new route. That is, the gateway is changed to the source of the current message. This heuristic is optional.

Any entry that fails these tests is ignored, as it is no better than the current route.

3.5. Output Processing

This section describes the processing used to create response messages that contain all or part of the routing table. This processing may be triggered in any of the following ways:

- by input processing when a request is seen. In this case, the resulting message is sent to only one destination.

- by the regular routing update. Every 30 seconds, a response containing the whole routing table is sent to every neighboring gateway. (See section 3.3.)

- by triggered updates. Whenever the metric for a route is changed, an update is triggered. (The update may be delayed; see below.)

Before describing the way a message is generated for each directly-connected network, we will comment on how the destinations are chosen for the latter two cases. Normally, when a response is to be sent to all destinations (that is, either the regular update or a triggered update is being prepared), a response is sent to the host at the opposite end of each connected point-to-point link, and a response is broadcast on all connected networks that support broadcasting. Thus, one response is prepared for each directly-connected network and sent to the corresponding (destination or broadcast) address. In most cases, this reaches all neighboring gateways. However, there are some cases where this may not be good enough. This may involve a network that does not support broadcast (e.g., the ARPANET), or a situation involving dumb gateways. In such cases, it may be necessary to specify an actual list of neighboring hosts and gateways, and send a datagram to each one explicitly. It is left to the implementor to determine whether such a mechanism is needed, and to define how the list is specified.

Triggered updates require special handling for two reasons. First, experience shows that triggered updates can cause excessive loads on networks with limited capacity or with many gateways on them. Thus the protocol requires that implementors include provisions to limit the frequency of triggered updates. After a triggered update is sent, a timer should be set for a random time between 1 and 5 seconds. If other changes that would trigger updates occur before the timer expires, a single update is triggered when the timer expires, and the timer is then set to another random value between 1 and 5 seconds. Triggered updates may be suppressed if a regular update is due by the time the triggered update would be sent.

Second, triggered updates do not need to include the entire routing table. In principle, only those routes that have changed need to be included. Thus messages generated as part of a triggered update must include at least those routes that have their route change flag set. They may include additional routes, or all routes, at the discretion of the implementor; however, when full routing updates require multiple packets, sending all routes is strongly discouraged. When a triggered update is processed, messages should be generated for every directly-connected network. Split horizon processing is done when generating triggered updates as well as normal updates (see below).

If, after split horizon processing, a changed route will appear identical on a network as it did previously, the route need not be sent; if, as a result, no routes need be sent, the update may be omitted on that network. (If a route had only a metric change, or uses a new gateway that is on the same network as the old gateway, the route will be sent to the network of the old gateway with a metric of infinity both before and after the change.) Once all of the triggered updates have been generated, the route change flags should be cleared.

If input processing is allowed while output is being generated, appropriate interlocking must be done. The route change flags should not be changed as a result of processing input while a triggered update message is being generated.

The only difference between a triggered update and other update messages is the possible omission of routes that have not changed. The rest of the mechanisms about to be described must all apply to triggered updates.

Here is how a response datagram is generated for a particular directly-connected network:

The IP source address must be the sending host's address on that network. This is important because the source address is put into routing tables in other hosts. If an incorrect source address is used, other hosts may be unable to route datagrams. Sometimes gateways are set up with multiple IP addresses on a single physical interface. Normally, this means that several logical IP networks are being carried over one physical medium. In such cases, a separate update message must be sent for each address, with that address as the IP source address.

Set the version number to the current version of RIP. (The version described in this document is 1.) Set the command to response. Set the bytes labeled "must be zero" to zero. Now start filling in entries.

To fill in the entries, go down all the routes in the internal routing table. Recall that the maximum datagram size is 512 bytes. When there is no more space in the datagram, send the current message and start a new one. If a triggered update is being generated, only entries whose route change flags are set need be included.

See the description in Section 3.2 for a discussion of problems raised by subnet and host routes. Routes to subnets will be meaningless outside the network, and must be omitted if the destination is not on the same subnetted network; they should be

replaced with a single route to the network of which the subnets are a part. Similarly, routes to hosts must be eliminated if they are subsumed by a network route, as described in the discussion in Section 3.2.

If the route passes these tests, then the destination and metric are put into the entry in the output datagram. Routes must be included in the datagram even if their metrics are infinite. If the gateway for the route is on the network for which the datagram is being prepared, the metric in the entry is set to 16, or the entire entry is omitted. Omitting the entry is simple split horizon. Including an entry with metric 16 is split horizon with poisoned reverse. See Section 2.2 for a more complete discussion of these alternatives.

3.6. Compatibility

The protocol described in this document is intended to interoperate with routed and other existing implementations of RIP. However, a different viewpoint is adopted about when to increment the metric than was used in most previous implementations. Using the previous perspective, the internal routing table has a metric of 0 for all directly-connected networks. The cost (which is always 1) is added to the metric when the route is sent in an update message. By contrast, in this document directly-connected networks appear in the internal routing table with metrics equal to their costs; the metrics are not necessarily 1. In this document, the cost is added to the metrics when routes are received in update messages. Metrics from the routing table are sent in update messages without change (unless modified by split horizon).

These two viewpoints result in identical update messages being sent. Metrics in the routing table differ by a constant one in the two descriptions. Thus, there is no difference in effect. The change was made because the new description makes it easier to handle situations where different metrics are used on directly-attached networks.

Implementations that only support network costs of one need not change to match the new style of presentation. However, they must follow the description given in this document in all other ways.

4. Control functions

This section describes administrative controls. These are not part of the protocol per se. However, experience with existing networks suggests that they are important. Because they are not a necessary part of the protocol, they are considered optional. However, we strongly recommend that at least some of them be included in every

implementation.

These controls are intended primarily to allow RIP to be connected to networks whose routing may be unstable or subject to errors. Here are some examples:

It is sometimes desirable to limit the hosts and gateways from which information will be accepted. On occasion, hosts have been misconfigured in such a way that they begin sending inappropriate information.

A number of sites limit the set of networks that they allow in update messages. Organization A may have a connection to organization B that they use for direct communication. For security or performance reasons A may not be willing to give other organizations access to that connection. In such cases, A should not include B's networks in updates that A sends to third parties.

Here are some typical controls. Note, however, that the RIP protocol does not require these or any other controls.

- a neighbor list - the network administrator should be able to define a list of neighbors for each host. A host would accept response messages only from hosts on its list of neighbors.

- allowing or disallowing specific destinations - the network administrator should be able to specify a list of destination addresses to allow or disallow. The list would be associated with a particular interface in the incoming or outgoing direction. Only allowed networks would be mentioned in response messages going out or processed in response messages coming in. If a list of allowed addresses is specified, all other addresses are disallowed. If a list of disallowed addresses is specified, all other addresses are allowed.

REFERENCES and BIBLIOGRAPHY

[1] Bellman, R. E., "Dynamic Programming", Princeton University Press, Princeton, N.J., 1957.

[2] Bertsekas, D. P., and Gallaher, R. G., "Data Networks", Prentice-Hall, Englewood Cliffs, N.J., 1987.

[3] Braden, R., and Postel, J., "Requirements for Internet Gateways", USC/Information Sciences Institute, RFC-1009, June 1987.

[4] Boggs, D. R., Shoch, J. F., Taft, E. A., and Metcalfe, R. M., "Pup: An Internetwork Architecture", IEEE Transactions on Communications, April 1980.

[5] Clark, D. D., "Fault Isolation and Recovery," MIT-LCS, RFC-816, July 1982.

[6] Ford, L. R. Jr., and Fulkerson, D. R., "Flows in Networks", Princeton University Press, Princeton, N.J., 1962.

[7] Xerox Corp., "Internet Transport Protocols", Xerox System Integration Standard XSIS 028112, December 1981.